(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平11－312088

(43)公開日　平成11年(1999)11月９日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/44	５５２	Ｇ０６Ｆ	9/44	５５２
	9/46	３６０		9/46	３６０Ｂ
	15/16	６２０		15/16	６２０Ｗ

審査請求　有　　請求項の数９　ＯＬ　（全 17 頁）

(21)出願番号　特願平11－32596

(22)出願日　平成11年(1999)２月10日

(31)優先権主張番号　０９／０３６２７０

(32)優先日　1998年３月６日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　ジェオフリィ・アレクサンダー・コーヘン
アメリカ合衆国　27705　ノースカロライナ州　ダルハム　ピンナクル　ロード　８

(74)代理人　弁理士　坂口　博　（外１名）

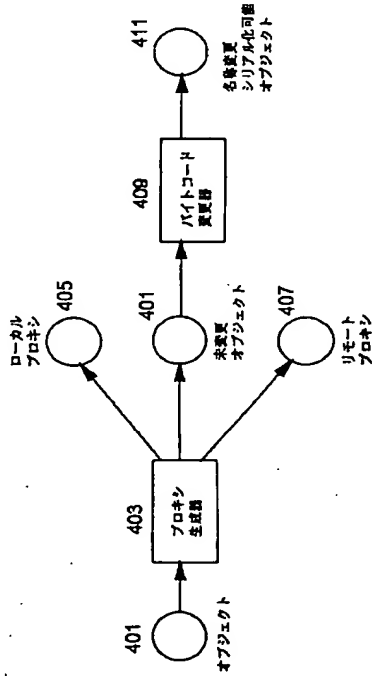
最終頁に続く

(54)【発明の名称】　プログラムの配布方法およびシステム

(57)【要約】

【課題】　プログラマの介在なしにプログラムをダイナミックに再構成できるようにする方法およびシステムを提供する。

【解決手段】　システム上で実行しているプログラムのソースコードを変更することなく、コンピュータ・ネットワーク内の複数のコンピュータ間に、プログラムをダイナミックに配布することができる。さらに、この方法およびシステムは、システムの管理者が、ダイナミックに再構成されるプログラムのソーステキストを変更することなしに、再構成が行われる条件を指定できることを可能にする。



1

【特許請求の範囲】

【請求項１】プログラムのオブジェクトを、第１のコンピュータから１つ以上のリモート・コンピュータへダイナミックに配布する方法であって、前記オブジェクトの各々は１つ以上のプログラム化エンティティを有し、

前記プログラム内のすべてのオブジェクトを識別するステップと、

前記プログラムのオブジェクトが移行される条件のリストを作成するステップと、

移行できる各オブジェクトについて、オブジェクトの外部からアクセスされる各前記プログラム化エンティティを識別するステップと、

リモート・コンピュータに移動でき、リモート・コンピュータからアクセスできる各オブジェクトについて、第１のプロキシおよび第２のプロキシを生成するステップとを含み、前記第１のプロキシは、前記最初のオブジェクトと同じ物理的デバイス上にあり、前記第２のプロキシは、前記同じ物理的デバイス上で生成されて、オブジェクト移行時に、前記リモート・コンピュータに転送され、前記第１のプロキシは、ネットワーク・リンケージおよび指示を含み、前記リモート・コンピュータ上の前記プログラム化エンティティをアクセスし、前記第２のプロキシは、ネットワーク・リンケージおよび指示を含み、前記第１のコンピュータ上の前記プログラム化エンティティをアクセスし、

前記条件のリストの前記オブジェクトの対応する条件が合うときに、前記第１のコンピュータからリモート・コンピュータへ、オブジェクトを移動するステップを含む、ことを特徴とする方法。

【請求項２】前記プログラム内のすべてのオブジェクトのうちのいずれが、移行のために利用できるかを識別するステップと、

移行のために利用できるオブジェクトの名称を、リストに配置するステップと、

前記リストを、配布方法への入力として利用するステップと、を含むことを特徴とする請求項１記載の方法。

【請求項３】前記第１のコンピュータからリモート・コンピュータへオブジェクトを移動するステップは、

前記オブジェクトのコピーを、前記リモート・コンピュータに移動するステップと、

前記第２のプロキシを、前記リモート・コンピュータへ移動するステップと、

前記リモート・コンピュータ上の前記第２のプロキシを呼出すために、前記第１のプロキシを送るステップと、を含むことを特徴とする請求項１または２記載の方法。

【請求項４】プログラムのオブジェクトを、第１のコンピュータから１つ以上のリモート・コンピュータへダイナミックに配布するコンピュータ読取り可能なコードであって、前記オブジェクトの各々は、１つ以上のプログラム化エンティティを有し、

2

前記プログラム内のすべてのオブジェクトを識別する第１のサブプロセスと、

前記プログラムのオブジェクトが移行される条件のリストを作成する第２のサブプロセスと、

前記第１および第２のサブプロセスに応答し、移行できる各オブジェクトについて、オブジェクトの外部からアクセスされる各前記プログラム化エンティティを識別するサブプロセスと、

前記第１および第２のサブプロセスに応答し、リモート・コンピュータに移動でき、リモート・コンピュータからアクセスできる各オブジェクトについて、第１のプロキシおよび第２のプロキシを生成するサブプロセスとを含み、前記第１のプロキシは、前記最初のオブジェクトと同じ物理的デバイス上にあり、前記第２のプロキシは、前記同じ物理的デバイス上に生成されて、オブジェクト移行時に、前記リモート・コンピュータに転送され、前記第１のプロキシは、ネットワーク・リンケージおよび指示を含み、前記リモート・コンピュータ上の前記プログラム化エンティティをアクセスし、前記第２のプロキシは、ネットワーク・リンケージおよび指示を含み、前記第１のコンピュータ上の前記プログラム化エンティティをアクセスし、

前記条件のリストの前記オブジェクトの対応する条件が合うときに、前記第１のコンピュータからリモート・コンピュータへ、オブジェクトを移動するサブプロセスを含む、ことを特徴とするコンピュータ読取り可能なコード。

【請求項５】前記第１のサブプロセスに応答し、前記プログラム内のすべてのオブジェクトのうちのいずれが、移行のために利用できるかを識別するサブプロセスと、

移行のために利用できるオブジェクトの名称を、リストに配置するサブプロセスと、

前記リストを、配布方法への入力として利用するサブプロセスと、を含むことを特徴とする請求項４記載のコンピュータ読取り可能なコード。

【請求項６】前記第１のコンピュータからリモート・コンピュータへオブジェクトを移動するサブプロセスは、

前記オブジェクトのコピーを、前記リモート・コンピュータに移動するサブプロセスと、

前記第２のプロキシを、前記リモート・コンピュータへ移動するサブプロセスと、

前記リモート・コンピュータ上の前記第２のプロキシを呼出すために、前記第１のプロキシを送るサブプロセスと、を含むことを特徴とする請求項４または５記載のコンピュータ読取り可能なコード。

【請求項７】通信機構を用いて共に接続された複数のコンピュータを有するコンピュータ・ネットワーク内に含まれるシステムであって、前記システムは、プログラムのオブジェクトを、第１のコンピュータから１つ以上のリモート・コンピュータへダイナミックに配布する方法

を含み、前記オブジェクトの各々は、１つ以上のプログラム化エンティティを有し、
前記プログラム内のすべてのオブジェクトを識別する手段と、
前記プログラムのオブジェクトが移行される条件のリストを作成する手段と、
移行できる各オブジェクトについて、オブジェクトの外部からアクセスされる各前記プログラム化エンティティを識別する手段と、
リモート・コンピュータに移動でき、リモート・コンピュータからアクセスできる各オブジェクトについて、第１のプロキシおよび第２のプロキシを生成する手段とを備え、前記第１のプロキシは、前記最初のオブジェクトと同じ物理的デバイス上にあり、前記第２のプロキシは、前記同じ物理的デバイス上に生成されて、オブジェクト移行時に、前記リモート・コンピュータに転送され、前記第１のプロキシは、ネットワーク・リンケージおよび指示を含み、前記リモート・コンピュータ上の前記プログラム化エンティティをアクセスし、前記第２のプロキシは、ネットワーク・リンケージおよび指示を含み、前記第１のコンピュータ上の前記プログラム化エンティティをアクセスし、
前記条件のリストの前記オブジェクトの対応する条件が合うときに、前記第１のコンピュータからリモート・コンピュータへ、オブジェクトを移動する手段を備える、ことを特徴とするシステム。
【請求項８】前記プログラム内のすべてのオブジェクトのうちのいずれが、移行のために利用できるかを識別する手段と、
移行のために利用できるオブジェクトの名称を、リストに配置する手段と、
前記リストを、配布方法への入力として利用する手段と、を備えることを特徴とする請求項７記載のシステム。
【請求項９】前記第１のコンピュータからリモート・コンピュータへオブジェクトを移動する手段は、
前記オブジェクトのコピーを、前記リモート・コンピュータに移動する手段と、
前記第２のプロキシを、前記リモート・コンピュータへ移動する手段と、
前記リモート・コンピュータ上の前記第２のプロキシを呼出すために、前記第１のプロキシを送る手段と、を有することを特徴とする請求項７または８記載のシステム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】オブジェクトをコンピュータからコンピュータへダイナミックに移動する能力は有用であることは、長い間知られていた。例えば、オブジェクトを移動することは、システムがコンピュータ間にバランスを与えることを可能にする。さらに、オブジェクトを移動することは、システムがしばしば通信するオブジェクトをダイナミックに集合させ、ネットワーク・トラヒックを軽減し、システムの全性能を改善することを可能にする。

【０００２】
【従来の技術】しかし、オブジェクトが再配置される（移行される）ことを可能にする現在のシステムは、重大な欠点に悩まされており、典型的に、専用プログラミング言語（または汎用言語への専用変更）、または専用オペレーティング・システムを必要とする。前者の場合、プログラマは新しいスキルを開発し、それらのコードを追加の構文で修飾しなければならない。後者の場合、得られるプログラムは、たまにしか利用されないシステム上での実行が制限される。
【０００３】ダイナミック・オブジェクト配布（Ｄｙｎａｍｉｃ　Ｏｂｊｅｃｔ　Ｄｉｓｔｒｉｂｕｔｉｏｎ：ＤＯＤ）に関係する本発明は、これらの制限のいずれにも悩まされることのないシステムを開示する。好適な実施例では、プログラマは、プログラムを、特殊な構文または表記を付加することなしに、Ｊａｖａプログラミング言語（Ｊａｖａは、Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓの登録商標である）で簡単に書くことができる。そしてプログラムを、産業界で広く利用されている標準的なＪａｖａ仮想マシーン（Ｊａｖａ　Ｖｉｒｔｕａｌ　Ｍａｃｈｉｎｅ：ＪＶＭ）で実行する。
【０００４】オブジェクト移行は、大学および産業界で広く研究されてきた。ｖカーネルのようなシステムは、ｖカーネルを実行するシステム上でのページベースの移行を可能にするが、ｖカーネルを有さないシステムは、移行を行うことができない。ｖカーネルは、オペレーティング・システム変更を要求するので、広くは用いられていない。ワシントン大学のＡｍｂｅｒのようなシステムは、マルチプロセッサ・コンピュータのノード間で、オブジェクトを移行する。しかし、Ａｍｂｅｒは、プログラムを特殊な言語で書くことを必要とする。Ａｍｂｅｒの実行時間は、また、広く用いられていない。本発明の主な利点は、カーネルの変更を必要とせず、広く利用されている言語であるＪａｖａで働くことである。
【０００５】本発明に関連する出願としては、以下のものがある。
米国特許第０８／８５２，２６３号明細書“Ａ　Ｐｒｏｃｅｓｓ　ｆｏｒ　Ｒｕｎｎｉｎｇ　Ｏｂｊｅｃｔｓ　Ｒｅｍｏｔｅｌｙ”
米国特許第０８／９４１，８１５号明細書“Ａｐｐａｒａｔｕｓ　ａｎｄ　Ｍｅｔｈｏｄ　ｆｏｒ　Ｄｙｎａｍｉｃａｌｌｙ　Ｍｏｄｉｆｙｉｎｇ　ＣｌａｓｓＦｉｌｅｓ　Ｄｕｒｉｎｇ　Ｌｏａｄｉｎｇ　ｆｏｒ　Ｅｘｅｃｕｔｉｏｎ”
【０００６】ＤＯＤシステムは、オブジェクトについて

5

のローカルおよびリモート“プロキシ（ｐｒｏｘｉｅｓ）”を、バイトコード・ファイルから生成する。プロキシは、オブジェクトへの、およびオブジェクトからのメソッド・コールを代行受信し、コールをオブジェクトへ送る。オブジェクトがローカルであると（オブジェクトは移行されなかった）、ローカル・プロキシは、コールをオブジェクトへ直接に送り、オブジェクトがリモートであると（オブジェクトは移行された）、ローカル・プロキシは、コールをリモート・プロキシに送り、リモート・プロキシがコールをオブジェクトに送る。バイトコード変更ツールは、オブジェクトの名称と、オブジェクトへのリファレンスを調整し、オブジェクトとプロキシとの間で名称競合が発生しないことを保証する。

【０００７】図１は、背景技術として、標準Ｊａｖａオブジェクト上のメソッドから、他の標準Ｊａｖａオブジェクトへのコール、つまり図１ではオブジェクトＡ１０１からオブジェクトＢ１０５へのコールを示している。図２および図３は、オブジェクトが移行した前後の典型的なコールのシナリオを示す。図２は、コールがローカル・プロキシ１０３を経てローカル・オブジェクト１０５に間接的に送られることを示している。図３は、コールが、ローカル・プロキシ１０３からリモート・プロキシ１０４へ、そしてリモート・マシーン上にあるオブジェクト１０６へ送られることを示している。

【０００８】ユーザ指定の条件（または前述した“述語（ｐｒｅｄｉｃａｔｅｓ））”に基づいて、“移行スレッド（ｍｉｇｒａｔｉｉｏｎ　ｔｈｒｅａｄ）”は、いつオブジェクトが移行されるべきかを決定する。“移行スレッド”は、コンピュータ上の資源をモニタして、いつ述語が満足されるかを決定する。（任意に、移行スレッドは、また、リモート・コンピュータについての情報を収集する。例えば、移行スレッドは、標準Ｕｎｉｘのコマンド“ｒｕｐｔｉｍｅ”を用いてこのような情報を収集することができる。このコマンドは、リモート・コンピュータのプロセッサ負荷についての情報を戻す。このような情報は、移行の決定に用いることができる。）オブジェクトを移行するためには、移行スレッドは、ローカル・プロキシを命令して、メソッド・コールをリモート・プロキシへ送り、およびオブジェクトのシリアル化されたバージョンおよびそのリモート・プロキシをリモート・コンピュータへ送る。リモート・コンピュータでは、両方のオブジェクトは、アンシリアル化される。シリアル化は、図１１に示され、さらにＪａｖａｉｎａ Ｎｕｔｓｈｅｌｌ，ｓｅｃｏｎｄ ｅｄｉｔｉｏｎ，Ｏ‘Ｒｅｉｌｌｙ，Ｍａｙ　１９９７，ｐｐ．１７２－１７７に説明されている。シリアル化は、オブジェクトのフラット化であり、したがってオブジェクトは、ネットワークを経て、より効率的に転送できる。そして、アンシリアライザは、目的地で元のオブジェクトを再構成する。

6

【０００９】

【発明が解決しようとする課題】本発明の目的は、プログラマの介在なしに、プログラムがダイナミックに再構成できるようにする方法を提供することにある。

【００１０】本発明の他の目的は、システム上で実行されているプログラムのソース・テキストに変更することなく、システムがダイナミックに再構成可能となるようにすることにある。

【００１１】本発明のさらに他の目的は、ダイナミックに再構成されるべきプログラムのソース・テキストに変更することなく、再構成が生じる条件を、管理者が特定できるようにすることにある。

【００１２】本発明のさらに他の目的は、ユーザベースで再構成を行う条件を、プログラマが指定できるようにすることにある。

【００１３】本発明のさらに他の目的は、配布の時にプログラマの介在なしに、システムをダイナミックかつ自動的に再構成することにある。

【００１４】

【課題を解決するための手段】本発明により定義されるダイナミック・オブジェクト配布（ＤＯＤ）は、入力として、次のものを有する。

―ソフトウェア・アプリケーションよりなるＪａｖａクラス・ファイルのセット。各クラス・ファイルは、Ｊａｖａバイトコードを含んでいる（Ｊａｖａは、Ｓｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓ，Ｉｎｃ．の登録商標である。）

―プログラマが、移行を可能にすることを選択するオブジェクトのリスト。デフォルトによって、ＤＯＤは、すべてのオブジェクトが移行可能であると仮定する。

―ＤＯＤがオブジェクトを移行する条件（“述語”）のリスト。

【００１５】

【発明の実施の形態】本発明（ＤＯＤ）は、自動オブジェクト配布（ＡＯＤ，前記した米国特許出願第０８／８５２，３６３号明細書に記載されている）に類似のプロキシ方式を用いている。したがって、ここでＡＯＤについては簡単に触れておく。ＡＯＤは、プログラマまたはシステム管理者が、プログラムが実行を開始する前のいつでも、プログラムをいかにクライアントおよびサーバのコンピュータに配布すべきかを可能にする。ＡＯＤは、アプリケーションが配布アプリケーションとして実行することを可能にするコードを自動的に生成する。

【００１６】しかし、前記したＡＯＤの発明について述べられているように、ＡＯＤは、オブジェクトが実行時間にクライアントとサーバとの間を移行することを許さない。ＡＯＤは、実行の前に、配布が完了することを要求する。実行時間の間にオブジェクトを移動して、サーバ負荷のような変化する状態に調整することは有益である。すなわち、サーバがよりビジーになると、より多く

のクラスがクライアントに移動し、サーバの負荷を軽減する。本発明では、実行時間中にオブジェクトが移動することを可能にするＡＯＤへの強化について開示する。

【００１７】まず、ＡＯＤを再検討するのに、以下の例を用いる。オブジェクト“ａ”は、クラスＡにより例示され、クラスＡは、クラスＢにより例示されるオブジェクト“ｂ”を含んでいる（関係がある）ものとする。クラスＢは、ｍｅｔｈｏｄ　ｆｏｏ（）を有している。この状況に対する擬似コードを以下に示す。

【００１８】

```
class A [
 A O[
  // constructor for A
 ]
 some$ method O [
  B b = new B O; // allocate object b

  b. foo O; // call for method on b
 ]
]
class B [
 B O[
  // constructor for B
]
 foo O [ // foo preform some action
 ]
]
```

プログラマが、“ａ”は“ｂ”から分割されるべきであると決定するならば、ＡＯＤプロセスは、クラスＢに対し２つのプロキシＢ′およびＢ″を生成するであろう。“ａ”から“ｂ”へのコールは、プロキシＢ′によって代行受信され、ネットワークを経てプロキシＢ″へ送られ、プロキシＢ″はクラスＢにローカル・コールを行い、結果を必要に応じて戻す。この構成が一度形成されると、構成は実行時間に変更できない。

【００１９】ダイナミック・オブジェクト配布（ＤＯＤ）は、ＡＯＤを強化するものである。ＤＯＤでは、プログラマは、どのクラスを最初にクライアントに与えるか、およびどのクラスを最初にサーバに与えるかのみならず、どのクラスを一方から他方へダイナミックに移動させるかを識別する。どのクラスを移動させるかについての識別は、プログラムが実行されるまではいつでも行うことができる。プログラマは、プログラム自体を変化させる（既知の技術であり、以下の説明を参照されたい）ことによってではなく、別個のファイルをリストに（例えば）タイプすることによって、これらのクラスを識別する。仕様の他のメソッドは、当業者には明らかであろう。実行時に、プログラマ指定の“述語”を用いて、オブジェクトの自動移行をトリガする。

【００２０】ＤＯＤプロセスは、以下のステップを含む。これらのステップ（本発明の好適な実施例としての）は、Ｊａｖａプログラミング言語および実行環境で表されるが、他のオブジェクト指向プログラミング・システムでは、他の実施例も可能である。

１）Ｊａｖａプログラミング言語のアプリケーションを有するクラスを書込む。

２）ソースＪａｖａファイルを、Ｊａｖａバイトコード・ファイルへコンパイルする。

３）どのオブジェクトが移動するのに利用できるかを識別する。

４）オブジェクトを移動できる述語（条件）を指定する。

５）システムの各クラスにより例示されたオブジェクトについて初期実行位置を指定する。

６）ローカルおよびリモート移行プロキシを生成する（以下に説明する）。

【００２１】次に、ユーザは、プログラムをスタートさせる。プログラムが実行されるにつれて、ＤＯＤプロセスは、ステップ４の述語が満足されたときに、オブジェクトを自動的に移動する。

【００２２】これらの各ステップに加えて、ＤＯＤは、“移行スレッド”を含んでいる。このスレッドは、システム資源をモニタする責任があり、述語（ステップ４に記述され、さらに以下に説明される）がいつ満足されたかを決定する。また、スレッドは、オブジェクト移行を開始させる。

【００２３】このプロセスを、次に、詳細に説明する。ステップ１および２は、Ｊａｖａについてのアプリケーション開発の標準部分である。従来の実施では、ステップ２の後に、ユーザによりアプリケーションが実行される。しかし、このアプリケーションは、オブジェクトをダイナミックに移動させる能力に欠けている。

【００２４】次に、開発者が“モバイル（ｍｏｂｉｌｅ）であることを望むクラスが識別される。モバイル・クラスにより例示されるオブジェクトは、ＤＯＤによる移行のための候補である。プログラマが、リストに入らないことを選択すると、ＤＯＤは、すべてのオブジェクトを移行できると仮定する。したがって、プログラマは、デフォルトによって、ソースの変更を行うことを必要とせず、さらに、オブジェクト移行を可能にするなんらの追加の作業を必要としない。これは、既知の技術から逸脱している。しかし以下に説明するように、可能な移行のためのオブジェクトを準備することは、システム・オーバヘッドを伴う。プログラマは、どのオブジェクトが移行されないかを知ることによって、このオーバヘッドを軽減することができる。プログラムに対し無変更を要求することは、別個のリストが必要とされるとしても、現在の技術に対してかなりの利点を与える。

【００２５】ステップ４では、述語が指定される。これらの述語を用いて、システムのダイナミック動作を制御

する。プログラマは、オブジェクトが移行される条件を識別する。例えば、Ｏ１～Ｏ９で示されるオブジェクトがサーバで実行されるならば、プログラマは、次のことを指定できる。すなわち、サーバの負荷が限界値Ｔ１を越えると、ＤＯＤがオブジェクトＯ１およびＯ２を移行させ、サーバの負荷が限界値Ｔ２を越えると、ＤＯＤがオブジェクトＯ３，Ｏ４，Ｏ５を移行させ、サーバの負荷が限界値Ｔ３を越えると、ＤＯＤがオブジェクトＯ６を移行させる。オブジェクトＯ７，Ｏ８，Ｏ９は移行されない。好適な実施例では、この情報は、“移行ファイル（ｍｉｇｒａｔｉｏｎ　ｆｉｌｅ）”と呼ばれるファイルにおいて簡単に指定される。しかし、他の仕様メソッドは、明らかに可能であり、当業者には自明であろう。

【００２６】移行ファイルは、ユーザ・ベースおよびよりグローバル・ベースで指定できることに留意すべきである。すなわち、好適な実施例では、移行ファイルの各バージョン（あるいは、１つの移行ファイル内の各セクション）は、アプリケーションの１つ以上のユーザに関係している。したがって、アプリケーションは、異なるユーザにより実行されるときに、異なる振舞いを任意に示すことができる。

【００２７】ステップ５は、システムの初期構成を示す。すなわち、プログラマは、どのクラスがサーバ・オブジェクトを生成するか、およびどのクラスがクライアント・オブジェクトを生成するかを、システムに知らせる。好適な実施例では、この情報は、移行ファイルにも格納される。

【００２８】移行ファイルの例は、次のとおりである。

```
Userl MyClassOne   CPU>.5 # move if CPU over half utilized
*Userl MyClassTwo   CPU>.7 # move if CPU over 70% utilized
Userl MyClassThree  CPU >.9 # move if CPU over 90 % utilized
Userl MyIOClassOne Disk <.4 # move if disk is 60% full
Userl MyIOClassTwo Disk <.2 # move if disk is 80% full
```

（“#”は、コメントの開示を示し、ライン上の次のテキストは無視される。）

ローカルおよびリモート・プロキシの生成は、ＤＯＤシステムに対する１つのキーである。これは例によって説明される。前に示した変更バージョンの例へ戻ることは、オブジェクト“ａ”および“ｂ”は、共に存在し、およびプログラマ指定のオブジェクトが、移行候補としてクラスＢにより例示されたものと仮定する。ＤＯＤは、クラスＢについてのバイトコード（“ｂ”は、インスタンスである）を読取り、その共用（ｐｕｂｌｉｃ）メソッドのすべてを調べる。ＡＯＤにおけるように、ＤＯＤは、Ｂ′と呼ばれるクラスＢのプロキシを生成する。したがって、クラスＢへのすべてのコールは、プロキシＢ′を経て間接的に送られる。ローカル・コールの場合には、呼出し順序は、Ａ→Ｂ′→Ｂである。リモート・コールの場合には、ＡＯＤにおけるように、呼出し順序は、Ａ→Ｂ′→Ｂ″→Ｂである。

【００２９】したがって、プロキシＢ′は、ローカル・コールおよびリモート・コールの両方を許容するように構成される。例示の擬似コードでは、プロキシＢ′は、以下のように表される。

【００３０】

```
class B[
 boolean local = TRUE; // object starts local
 B () [ // constructor for the proxy for B
 create a reference to the real B
 ]
 foo () [
   if (local) make local call to the local foo();
   else make remote call to foo as described in AOD
 ]
]
```

以下に示すように、各プロキシに対し、オブジェクトを移行させるためには、コードを付加しなければならない。さらに、タイミングの競合に対してメソッド・コールを保護するコードを付加しなければならない。すなわち、オブジェクトにおけるメソッドに対するコールは、オブジェクトが移行されている間は、許可されない。これは、プロキシ・クラスを“同期化（ｓｙｎｃｈｒｏｎｉｚｅｄ）”（標準的なＪａｖａ用語）させ、競争状態に対してクラスを保護し、移行メソッドを付加することによって行われる。最後に、ローカル・プロキシは、移行スレッドと共に自身を登録しなければならない。このことは、オブジェクトが移行されなければならないときに、移行スレッドが、プロキシを配置することを可能にする。

【００３１】各プロキシは、“移行可能（Ｍｉｇｒａｔａｂｌｅ）”インタフェースを実行するように、構成される。このことは、移行スレッドの“登録（ｒｅｇｉｓｔｅｒ）”メソッドが、単一タイプのオブジェクトを、

11

パラメータすなわち移行インタフェースを実行するプロキシ・クラスとして、受取ることを可能にする。したがって移行スレッドの登録メソッドは、オブジェクトが移行されなければならないときに用いるテーブルに、パラメータ（プロキシ・オブジェクト）へのリファレンスを

12

*格納する。

【００３２】したがってプロキシB′は、次のようになる。

【００３３】

```
synchronized class B implements migratable [
 boolean local = TRUE; // object starts local
 B() [ // constructor for the proxy for B
 create a reference to the real B
 register with the migration thread
 ]
 migrate () [
  serialize the "real" b;
  send the serialized object via a socket to the partner where it
    will be instantiated
  local = FALSE
 ]
 foo () [
   if (local) make local call to the "real" foo ();
 else make remote call to foo as described in AOD
 ]
]
```

プロキシ・オブジェクトB′は、標準Ｊａｖａの意味構造によって同期されるので、移行メソッドは、オブジェクトにおける他のメソッドが呼出されている間は、実行されず、同様に、オブジェクトが移行している間は、メソッドは呼出すことができないことに留意すべきである。

【００３４】オブジェクトのシリアル化、およびシリアル化されたオブジェクトをソケットを通過させることは、Ｊａｖａにおいて周知の技術である。オブジェクトがシリアル化されて、ネットワーク上に送られた後（“ライトオブジェクト（ｗｒｉｔｅＯｂｊｅｃｔ）”と呼ばれる標準的なＪａｖａメソッドを用いて）、オブジェクトは、他の標準Ｊａｖａメソッド（“リードオブジェクト（ｒｅａｄＯｂｊｅｃｔ）”）を用いて、リモート・コンピュータ上でＤＯＤコンポーネントによって受け取られる。そして、アンシリアル化されたオブジェクトが、リモート・コンピュータ上で実行される準備状態にある。

【００３５】典型的に、シリアル化されるためには、標準Ｊａｖａのリリースに設けられたシリアル化可能（Ｓｅｒｉａｌｉｚａｂｌｅ）インタフェースをクラスが実行しなければならない。しかし、シリコン化可能インタフェースを実行しないクラスの例は、ＪＶＭによってシリアル化されない。しかし、ＤＯＤは、ソース変更なしに、オブジェクトが移行できることを要求し、およびＪａｖａは、シリアル化可能オブジェクトが、シリアル化可能インタフェースを実行することを要求する。

【００３６】この明らかな問題は、ユーザによってモバイルとしてリストされているが、シリアル化可能インタフェースを実施しないこれらのクラスを識別することによって解決される。前述した米国特許出願“Ａｐｐａｒａｔｕｓ ａｎｄ Ｍｅｔｈｏｄ ｆｏｒ Ｄｙｎａｍｉｃａｌｌｙ Ｃｌａｓｓ Ｆｉｌｅｓ Ｄｕｒｉｎｇ Ｌｏａｄｉｎｇ ｆｏｒ Ｅｘｅｃｕｔｉｏｎ”に記載されているＪＯＩＥツールのような既知のバイトコード変更ツールを用いて、ＪＶＭによってクラスが最初に負荷されるときの時点で、クラスをシリアル化可能インタフェースの実行としてマークする変換が与えられる。この時点では、実行は通常通り進行する。変更（クラスにシリアル化可能インタフェースを実行させる）は、バイトコード変更ツールによって、自動的に行われるので、プログラマは何かをすることは要求されず、ソースコードに対する変更は生じない。同様のメカニズムを用いて、パラメータ・オブジェクトが、ローカル・プロキシからリモート・プロキシへ転送されなければならないならば、パラメータ・オブジェクトは、シリアル化可能インタフェースを実行するように作成することができる。図９は、プロキシ生成およびバイトコード変更のプロセスを示す。

【００３７】図９において、オブジェクト４０１は、プロキシ・ジェネレータ４０３によって処理される。プロキシ生成器は、ローカル・プロキシ４０５とリモート・プロキシ４０７を生成し、および未変更のオブジェクト４０１を保持する。バイトコード変更器４０９は、未変更オブジェクト４０１を処理して、これからシリアル化可能オブジェクト４１１を生成する。

13

【００３８】ＡＯＤにおけるように、リモート・プロキシ（この例ではＢ″）は、ローカル・プロキシからのコールを受取り、ローカル・コールを元のオブジェクト（Ｂ）に対して行う。さらに、リモート・プロキシは、リモート・オブジェクトから、リモート・プロキシを経て、およびローカル・プロキシを経て、最後に呼出し先へコールを転送するためのコードを含まなければならない。

【００３９】図１０に示すように、リターン・コールのためのコードを生成するプロセスは、ＡＯＤにおいて用いられるプロセス、および着信コールのためにここで用いられるプロセスに非常に類似している。移行可能なオブジェクトのためのバイトコードは検査され、他のオブジェクトに対するすべてのメソッド・コールが取出される。このような各コールに対し、ローカル・プロキシ５１０およびリモート・プロキシ５２０の両方が構成され、これらはバイトコード内にあるメソッドに類似の名称のメソッドを含んでいる。リモート・プロキシにおけるコードは、メソッドが呼びだされるとき、ローカル・プロキシへのリモート（ＲＭＩ）コールが実行される。ローカル・プロキシにおけるコードは、リモート・コールを受信したとき、実際の呼出し先にローカル・コールを行う。したがって、このプロセスは、可能性のある着信コール（すなわち、共用メソッド）の代りに、オブジェクトが発信コールに対して検査されることを除いて、着信コールに対してプロキシを生成するプロセスとほぼ同じである。ローカル・プロキシおよびリモート・プロキシの通常の機能は、この場合反転されることに留意すべきである。

14

*【００４０】ローカル・マシンに戻るようにオブジェクトを移行させることは、プロセスの反転を事実上伴う。プロセスは、“非移行（ｕｎｍｉｇｒａｔｉｏｎ）”が発生することを検出すると、移行スレッドによって開始される。移行スレッドは、ローカル・プロキシにおける“非移行”メソッドを呼出す。

【００４１】非移行の際には、タイミング問題が発生しないことを保証することは重要である。ローカル・プロキシが同期されるので、ローカル・プロキシにおいて実行される唯一のメソッドとなる時まで、非移行メソッドへのコールを実行しない（すなわち、コールはブロックする）。リモート呼出し先へのすべてのコールが、ローカル・プロキシを通過しなければならないので、我々は、非移行が発生するとき、リモート呼出し先は実行しないことを知っている。ローカル・プロキシにおける非移行メソッドは、リモート・プロキシにおける移行メソッドを事実上呼出す。ローカル・プロキシにおける非移行メソッドへのコールは、全移行が終了するまで終わらない。

【００４２】リモート・プロキシにおける移行メソッドは、ローカル・プロキシにおける移行メソッドが実行する同じ機能を実行する。移行メソッドは、オブジェクトをシリアル化し、それをローカル・マシンへ転送する。ローカル・マシンは、オブジェクトをアンシリアル化する。この時点では、ローカル・プロキシは、その“ローカル”ブールをセットする。発展中の例では、プロキシＢ′は次のようになる。

【００４３】

```
synchronized class B[
 boolean local = TRUE; // object starts local
 B () [ // constructor for the proxy for B
 create a reference to the real B
 register with the migration thread
 ]
 unmigrate () [
  make a remote call to migrate on the remote proxy
  receive the serialized object
  unserialize the object
  local = TRUE;
  return;
 ]
 migrate () [
  serialize the "real" b;
  send the serialized object via a socket to the partner where it
    will be instantiated
  local = FALSE;
  return;
 ]
foo () [
```

15　　16

```
if (local) make local call to the "real" foo ();
else make remote call to foo as described in AOD
]
]
```

次に、ユーザは、クライアント／サーバＪａｖａアプリケーションであるときに、システムを実際にスタートさせる。アプリケーションが実行されると、ＤＯＤは、移行ファイルにおいて指定された述語をモニタする。述語が満足されると（例えば、負荷限界値が満足されると）、移行プロセスがトリガされる。

【００４４】移行を実行するためには、ＤＯＤプロセス

```
do forever [
 check for object registration
 check computer resource usage
 compare usages to predicates
 if a predicate is satisfied [
  check list of registered objects, initiating migration for
  those whose predicate was satisfied
 ]
]
```

ローカル・プロキシにおける移行メソッドは、オブジェクトをシリアル化し、オブジェクトはプロキシを有し、プロキシは移行メソッドを呼出して、実行のためのリモート・ノードに送る。リモート・マシン上のＪＶＭは、“リードオブジェクト”メソッドを呼出して、シリアル化されたオブジェクトを読取り、リモート・メソッド呼出し登録（Ｒｅｍｏｔｅ　Ｍｅｔｈｏｄ　Ｉｎｖｏｃａｔｉｏｎ（ＲＭＩ）Ｒｅｇｉｓｔｒｙ）で、リモート・プロキシを登録する。ＴＣＰ／ＩＰソケットを経てネットワークにわたるシリアル化されたオブジェクトを読取って書込む技術、およびＲＭＩ登録とで対話する技術は、周知である。さらに、移行されたオブジェクトについてのリモート・プロキシ（ステップ６で生成された）は、リモート・ノードに送られる。

【００４６】前述したローカル・プロキシは、代理しているオブジェクトによって含まれるすべての共用メソッドを含むように構成される。これらメソッドの各々は、実際のオブジェクトにおけるメソッド（実際のオブジェクトがローカルならば）を呼出し、あるいはリモート・オブジェクトにおけるメソッドをリモート的に（ＲＭＩを経て）呼出す（オブジェクトが移行され、現在リモートであるならば）。

```
class A [
 A () [
 // constructor for A
 ]
 some#method () [
  B b = new B (); // allocate object b
  b. foo (); // call for method on b
 ]
```

＊の移行スレッドが、述語が満足されたことを検出すると、移行ファイルを用いて、どのオブジェクトが移動すなわち移行されるかを決定する。これらは、各オブジェクトについてのプロキシにおける移行メソッド（前述した）を呼出す。移行スレッドは、以下に示される擬似コードを実行する。

【００４５】

※【００４７】移行可能なオブジェクトを呼出すオブジェクトについてのコードを変更することを避けるためには、プロキシは、プロキシとして働くオブジェクトの名称を持つようにする。したがって、移行可能なオブジェクトについて元々指定されたコールを、標準のＪａｖａ意味構造によって、プロキシで再目標とされる。

【００４８】しかし、Ｊａｖａは、２つのオブジェクトが、同一の名称を有することを許さない。オブジェクトがローカル的に実行されている場合には、プロキシおよびプロキシされるオブジェクトは、同一の名称を有する。この問題を是正するためには、名称変更プロセスが実行される。

【００４９】バイトコード変更ツールを用いて、元のオブジェクト（移行可能なオブジェクト）は、バイトコード・ファイルのその名称に任意のストリングを付加することによって、名称変更される。クラスの構成も、同様な方法により名称が変更される。次に、プロキシは、この新しい名称のオブジェクトに対し呼出しを行う。

【００５０】例えば、上記のことからコード・フラグメントを考察する。

【００５１】

```
17                                                    18
]
class B [
 B () [
 // constructor for B
 ]
 foo () [
   // foo performs some action
 ]
]
To create a proxy for B, first B is renamed in its bytecode ("class" f
ile), then
the proxy is created:
class Bwxyz [ // this is the real, renamed B
 Bwxyz () [
  // constructor for the real class
 ]
 foo () [
  // foo performs some action
 ]
]                                  20
synchronized calss B [ // this is the proxy for B
 Bwxyz myBwxyz; // reference to the original object
 B () [ // constructor for B
  myBwxyz = new Bwxyz (); // create a new object for the original B
  register with the migration thread
 ]
 foo () [ // calls foo on the renamed version of the original B
  myBwxyz. foo ();
 ]
 migrate () [ // as above, object migration code, called by migr. threa
d
  serialize the "real" b;
  send the serialized object via a socket to the partner where it
   will be instantiated
  local = FALSE;
 ]
 unmigrate () [
  make a remote call to migrate on the remote proxy
  receive the serialized object
  unserialize the object          40
  local = TRUE;
  return;
  ]
 ]
```

Aから“B．ｆｏｏ”へのコールは、プロキシ・オブジェクトBにおける“ｆｏｏ”メソッドに関係する。したがって、Bプロキシは、Bへの代行受信されたコールを有している。また、Bプロキシは、そのリファレンスを経て、元のオブジェクトBに呼出しを行う。

【００５２】図４は、３つのオブジェクトA，B，Cを含む簡単なオブジェクト指向プログラムを示す。オブジェクトAは、ａ１～ａ３で表示された３つのメソッドを有し、オブジェクトBは、ｂ１～ｂ４で表示された４つのメソッドを有し、オブジェクトCは、ｃ１，ｃ２で表示された２つのメソッドを有している。図に示すように、以下のメソッド・コールが存在する。

19

【００５３】
a1 calls b3 and b4
a2 calls b2
b2 calls c2
c1 calls a2
図５は、プログラマが、Ｃが唯一の移行可能なオブジェクトであることを指定した後の図４と同じプログラムを示す。システムは、図にＣＬとして指定されたローカル・プロキシを自動的に生成する。このＣＬは、ｃｌｌおよびｃｌ２で指定された２つのメソッドを含んでいる。命名の競合の議論について、ローカル・プロキシおよびそのメソッドは、オブジェクトおよびそのメソッドの元の名称で命名される。オリジナル・オブジェクトＣは、バイトコード変更ツールによって名称変更された（図にはオブジェクト名称変更は示されていない）。
【００５４】メソッド・コール・リストは、このように調整される。
【００５５】
a1 calls b3 and b4
a2 calls b2
b2 calls cl2
cl2 calls c2
cll calls cl (unused)
cl calls a2
これはローカル構成であるので、呼出し元（例えば、Ｂ）からのコールは、ローカル・プロキシ（ＣＬ）により代行受信され、標準的なメソッド・コールを経て呼出し先（Ｃ）へ送られる。
【００５６】図６は、移行スレッドを示す。このスレッドは、満足された述語を検出し、メッセージを、オブジェクトＣ（ＣＬ）についてのローカル・プロキシにおける“ｍｉｇｒａｔｅ（移行）”メソッドに送る。このことは、リモート・マシンへのオブジェクトＣの移行を生じさせ、および図７に示される構成に変化させる。
【００５７】図７は、オブジェクトＣがリモート・コンピュータへ移行した後の同じ構成操作を示す。メソッド・コール・リストは、次のようになる。
【００５８】
a1 calls b3 and b4
a2 calls b2
// Call from b2 to c2 indirected remotely
b2 calls cl2
cl2 calls cr2 using RMI
cr2 calls c2
// Potential, but unused, calls into cl
cll calls crl using RMI (unused)
crl calls cl (unused)
// Calls from cl to a2
cl calls ar2
ar2 calls all using RMI

20

al2 calls a2
前記したように、ＤＯＤは、以下のメインステップよりなる。すなわち、プログラマは、アプリケーションを書込み、コンパイルする。結果は、１組のＪａｖａバイトコード（クラス）ファイルである。これは、Ｊａｖａアプリケーションについての標準的な展開相であり、厳密には本発明の一部ではない。次に、プログラマは、どのオブジェクトを移行できるか、およびオブジェクトが移行すべき条件を識別する。次に、ＤＯＤは、ローカル・スタブおよびリモート・スタブを生成する。ＤＯＤは、バイトコード変更を実行して、適切なクラスがシリアル化可能インタフェースを実施し、すべての名称競合が解決されることを保証する。これらスタブは、メソッド・コールをオブジェクト（それがローカルであろうと、リモートであろうと）に送るためのコードを含んでいる。スタブはまた、オブジェクトを移行させるために必要なコードを含んでいる。次に移行スレッドは、システム資源をモニタし、述語がいつ満足されるかを決定する。述語が満足されると、述語はオブジェクトのプロキシにおける移行メソッドを呼出し、移行を生じさせる。次に、移行メソッドは、要求されるリモート・プロキシと共に、リモート・オブジェクトをシリアル化し、これらをリモートＪＶＭ（Ｊａｖａ仮想マシン）に転送する。リモートＪＶＭでは、オブジェクトが再例示される。次に、アプリケーションを、その新しい構成で再スタートすることができる。
【００５９】上述したＡＯＤの発明について、この移行は、メソッド・コール境界（これは、標準的な規則であり、良好なプログラミング・プラクティスである）に沿ったブレークを信頼するので、そのメンバ変数がアクセスされるオブジェクトは、直接に移行できない。困難性を避けるためには、潜在的に移行可能であるオブジェクトは、利用（ｐｒｉｖａｔｅ）メンバ変数のみを含まなければならない。
【００６０】まとめとして、本発明の構成に関して以下の事項を開示する。
（１）プログラムのオブジェクトを、第１のコンピュータから１つ以上のリモート・コンピュータへダイナミックに配布する方法であって、前記オブジェクトの各々は１つ以上のプログラム化エンティティを有し、前記プログラム内のすべてのオブジェクトを識別するステップと、前記プログラムのオブジェクトが移行される条件のリストを作成するステップと、移行できる各オブジェクトについて、オブジェクトの外部からアクセスされる各前記プログラム化エンティティを識別するステップと、リモート・コンピュータに移動でき、リモート・コンピュータからアクセスできる各オブジェクトについて、第１のプロキシおよび第２のプロキシを生成するステップとを含み、前記第１のプロキシは、前記最初のオブジェクトと同じ物理的デバイス上にあり、前記第２のプロキ

シは、前記同じ物理的デバイス上で生成されて、オブジェクト移行時に、前記リモート・コンピュータに転送され、前記第1のプロキシは、ネットワーク・リンケージおよび指示を含み、前記リモート・コンピュータ上の前記プログラム化エンティティをアクセスし、前記第2のプロキシは、ネットワーク・リンケージおよび指示を含み、前記第1のコンピュータ上の前記プログラム化エンティティをアクセスし、前記条件のリストの前記オブジェクトの対応する条件が合うときに、前記第1のコンピュータからリモート・コンピュータへ、オブジェクトを移動するステップを含む、ことを特徴とする方法。

（2）前記プログラム内のすべてのオブジェクトのうちのいずれが、移行のために利用できるかを識別するステップと、移行のために利用できるオブジェクトの名称を、リストに配置するステップと、前記リストを、配布方法への入力として利用するステップと、を含むことを特徴とする上記（1）に記載の方法。

（3）前記第1のコンピュータからリモート・コンピュータへオブジェクトを移動するステップは、前記オブジェクトのコピーを、前記リモート・コンピュータに移動するステップと、前記第2のプロキシを、前記リモート・コンピュータへ移動するステップと、前記リモート・コンピュータ上の前記第2のプロキシを呼出すために、前記第1のプロキシを送るステップと、を含むことを特徴とする上記（1）または（2）に記載の方法。

（4）プログラムのオブジェクトを、第1のコンピュータから1つ以上のリモート・コンピュータへダイナミックに配布するコンピュータ読取り可能なコードであって、前記オブジェクトの各々は、1つ以上のプログラム化エンティティを有し、前記プログラム内のすべてのオブジェクトを識別する第1のサブプロセスと、前記プログラムのオブジェクトが移行される条件のリストを作成する第2のサブプロセスと、前記第1および第2のサブプロセスに応答し、移行できる各オブジェクトについて、オブジェクトの外部からアクセスされる各前記プログラム化エンティティを識別するサブプロセスと、前記第1および第2のサブプロセスに応答し、リモート・コンピュータに移動でき、リモート・コンピュータからアクセスできる各オブジェクトについて、第1のプロキシおよび第2のプロキシを生成するサブプロセスとを含み、前記第1のプロキシは、前記最初のオブジェクトと同じ物理的デバイス上にあり、前記第2のプロキシは、前記同じ物理的デバイス上に生成されて、オブジェクト移行時に、前記リモート・コンピュータに転送され、前記第1のプロキシは、ネットワーク・リンケージおよび指示を含み、前記リモート・コンピュータ上の前記プログラム化エンティティをアクセスし、前記第2のプロキシは、ネットワーク・リンケージおよび指示を含み、前記第1のコンピュータ上の前記プログラム化エンティティをアクセスし、前記条件のリストの前記オブジェクト

の対応する条件が合うときに、前記第1のコンピュータからリモート・コンピュータへ、オブジェクトを移動するサブプロセスを含む、ことを特徴とするコンピュータ読取り可能なコード。

（5）前記第1のサブプロセスに応答し、前記プログラム内のすべてのオブジェクトのうちのいずれが、移行のために利用できるかを識別するサブプロセスと、移行のために利用できるオブジェクトの名称を、リストに配置するサブプロセスと、前記リストを、配布方法への入力として利用するサブプロセスと、を含むことを特徴とする上記（4）に記載のコンピュータ読取り可能なコード。

（6）前記第1のコンピュータからリモート・コンピュータへオブジェクトを移動するサブプロセスは、前記オブジェクトのコピーを、前記リモート・コンピュータに移動するサブプロセスと、前記第2のプロキシを、前記リモート・コンピュータへ移動するサブプロセスと、前記リモート・コンピュータ上の前記第2のプロキシを呼出すために、前記第1のプロキシを送るサブプロセスと、を含むことを特徴とする上記（4）または（5）に記載のコンピュータ読取り可能なコード。

（7）通信機構を用いて共に接続された複数のコンピュータを有するコンピュータ・ネットワーク内に含まれるシステムであって、前記システムは、プログラムのオブジェクトを、第1のコンピュータから1つ以上のリモート・コンピュータへダイナミックに配布する方法を含み、前記オブジェクトの各々は、1つ以上のプログラム化エンティティを有し、前記プログラム内のすべてのオブジェクトを識別する手段と、前記プログラムのオブジェクトが移行される条件のリストを作成する手段と、移行できる各オブジェクトについて、オブジェクトの外部からアクセスされる各前記プログラム化エンティティを識別する手段と、リモート・コンピュータに移動でき、リモート・コンピュータからアクセスできる各オブジェクトについて、第1のプロキシおよび第2のプロキシを生成する手段とを備え、前記第1のプロキシは、前記最初のオブジェクトと同じ物理的デバイス上にあり、前記第2のプロキシは、前記同じ物理的デバイス上に生成されて、オブジェクト移行時に、前記リモート・コンピュータに転送され、前記第1のプロキシは、ネットワーク・リンケージおよび指示を含み、前記リモート・コンピュータ上の前記プログラム化エンティティをアクセスし、前記第2のプロキシは、ネットワーク・リンケージおよび指示を含み、前記第1のコンピュータ上の前記プログラム化エンティティをアクセスし、前記条件のリストの前記オブジェクトの対応する条件が合うときに、前記第1のコンピュータからリモート・コンピュータへ、オブジェクトを移動する手段を備える、ことを特徴とするシステム。

（8）前記プログラム内のすべてのオブジェクトのうち

23

のいずれが、移行のために利用できるかを識別する手段と、移行のために利用できるオブジェクトの名称を、リストに配置する手段と、前記リストを、配布方法への入力として利用する手段と、を備えることを特徴とする上記（７）に記載のシステム。

（９）前記第１のコンピュータからリモート・コンピュータへオブジェクトを移動する手段は、前記オブジェクトのコピーを、前記リモート・コンピュータに移動する手段と、前記第２のプロキシを、前記リモート・コンピュータへ移動する手段と、前記リモート・コンピュータ上の前記第２のプロキシを呼出すために、前記第１のプロキシを送る手段と、を有することを特徴とする上記（７）または（８）に記載のシステム。

【図面の簡単な説明】

【図１】メソッド間のコールを示す図である。

【図２】移行前のオブジェクト内のメソッド間のコールを示す図である。

【図３】移行後のメソッド間のコールを示す図である。

【図４】３つのオブジェクトを有する標準オブジェクト指向プログラムを示す図である。

【図５】１つの再配置可能なオブジェクトを有するローカル構成を示す図である。

24

【図６】図５の構成についての移行スレッドを示す図である。

【図７】メソッドＣがリモート・ホストへ移行された後の動作を説明する図である。

【図８】メソッドＣがリモート・ホストへ移行された後の動作を説明する図である。

【図９】プロキシ生成およびバイトコードの生成を示す図である。

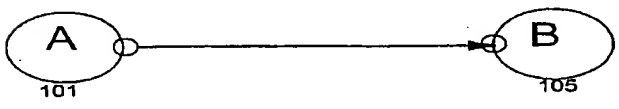
【図１０】コール・リターンの準備を示す図である。

【図１１】オブジェクトのシリアル化を示す図である。

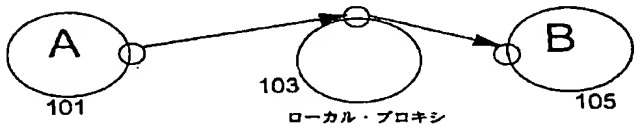
【符号の説明】

４０１　オブジェクト
４０３　プロキシ生成器
４０５　ローカル・プロキシ
４０７　リモート・プロキシ
４０９　バイトコード変更器
４１１　シリアル化可能オブジェクト
５１０　ローカル・プロキシ
５２０　リモート・プロキシ

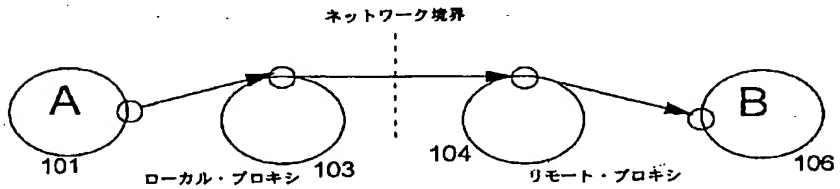
【図１】



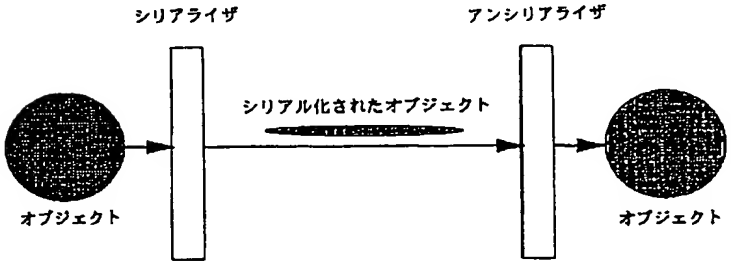
【図２】



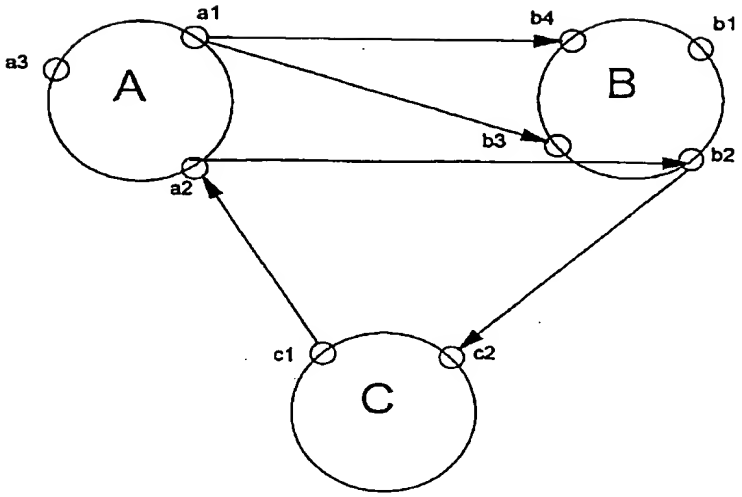
【図３】



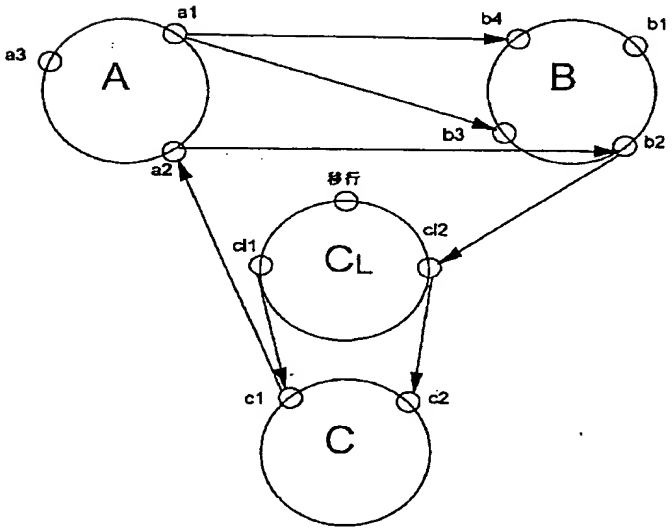
【図１１】



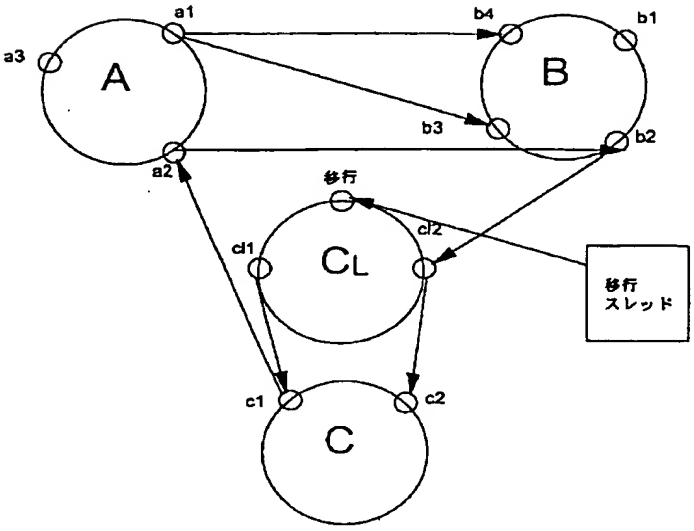
【図４】


a1
b4
b1
a3
A
B
b3
b2
a2
c1
c2
C

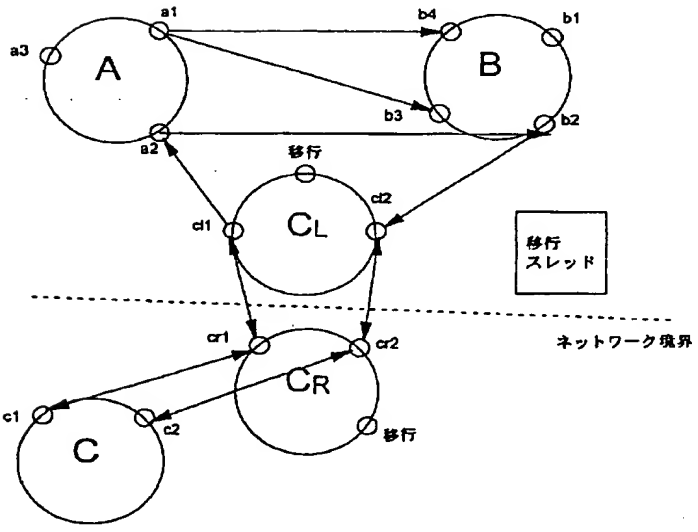
【図５】


a1
b4
b1
a3
A
B
b3
b2
a2
移行
cl1
cl2
CL
c1
c2
C

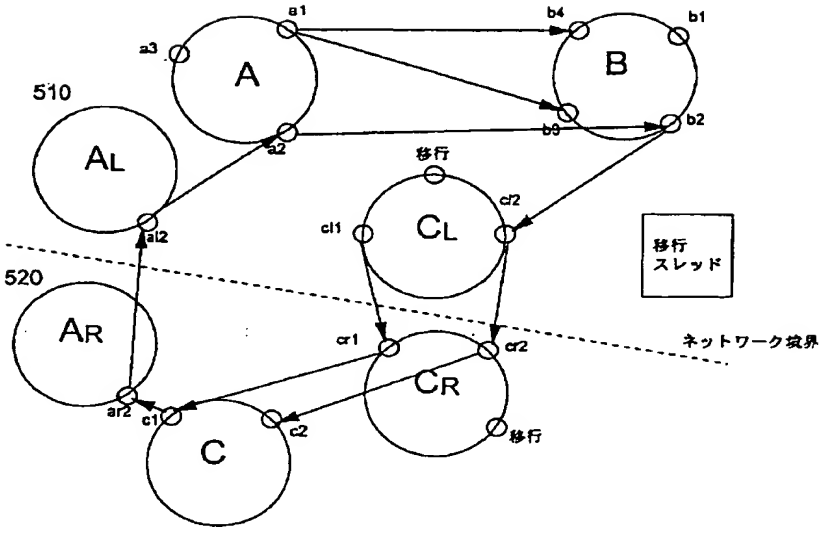
【図６】


a1
b4
b1
a3
A
B
b3
b2
a2
移行
cl2
cl1
CL
移行
スレッド
c1
c2
C

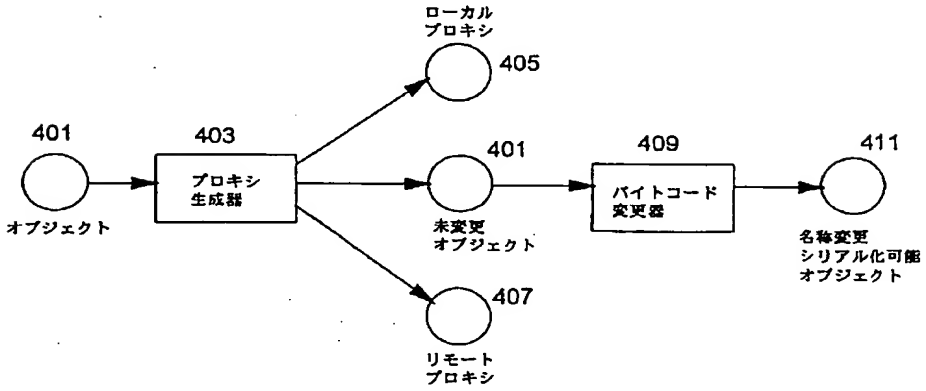
【図７】


a1
b4
b1
a3
A
B
b3
b2
a2
移行
cl2
cl1
CL
移行
スレッド
ネットワーク境界
cr1
cr2
CR
c1
c2
C
移行

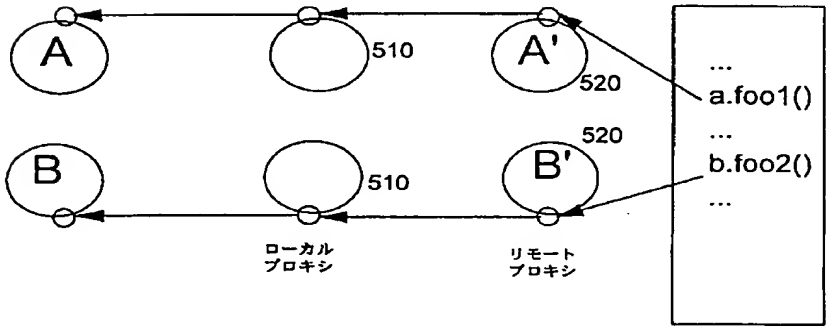
【図８】


a1
b4
b1
a3
A
B
510
AL
b3
b2
a2
移行
cl2
cl1
CL
移行
スレッド
al2
520
AR
ネットワーク境界
cr1
cr2
CR
ar2
c1
c2
C
移行

【図９】


ローカル
プロキシ
405
401
403
401
409
411
プロキシ
生成器
バイトコード
変更器
オブジェクト
未変更
オブジェクト
名称変更
シリアル化可能
オブジェクト
407
リモート
プロキシ

【図１０】


A
510
A'
520
...
a.foo1()
...
b.foo2()
...
520
B
510
B'
ローカル
プロキシ
リモート
プロキシ

フロントページの続き

(72)発明者　デヴィッド・ルイス・カミンスキィ
アメリカ合衆国　27514　ノースカロライナ州　チャペル　ヒル　コービン　ヒル　103

(72)発明者　リチャード・アダム・キング
アメリカ合衆国　27513　ノースカロライナ州　ケイリィ　エクスカリバー　コート　214